```
                                                        Program Segment
          ENTRY
          .                                          }
          .                                          }      1
          .                                          }
          .
          Test   X
          BC     EQ,Jump1         Branch  B1         }
          .                                          }
          .                                          }      2
          .                                          }
          .                                          }
          Test   X                                   }
          BC     EQ,Jump2         Branch  B2         }
          .                                          }
          .                                          }
          .                                          }
          Touch  A                Prefetch   A       }
          .                                          }      4
          .                                          }
          .                                          }
          Load   R5,A             Use        A       }
          .
          Return
  Jump1   *                                          }
          .                                          }      3
          .                                          }
          Test   X                                   }
          BC     EQ,Jump3         Branch  B3
          .                                          }
          .                                          }
          .                                          }
          Touch  C                Prefetch   C       }
          .                                          }      6
          .                                          }
          .                                          }
          Load   R5,C             Use        C       }
          .                                          }
          Return                                     }
  Jump2   *
          .                                          }
          .                                          }
          Touch  B                Prefetch   B       }
          .                                          }      5
          .                                          }
          .                                          }
          Load   B                Use        B       }
          .
          Return
  Jump3   *
          .                                          }
          .                                          }
          Touch  D                Prefetch   D       }
          .                                          }      7
          .                                          }
          .                                          }
          Load   D                Use        D       }
          .
          Return
```

Figure 1

Control Flow Graph Of Program

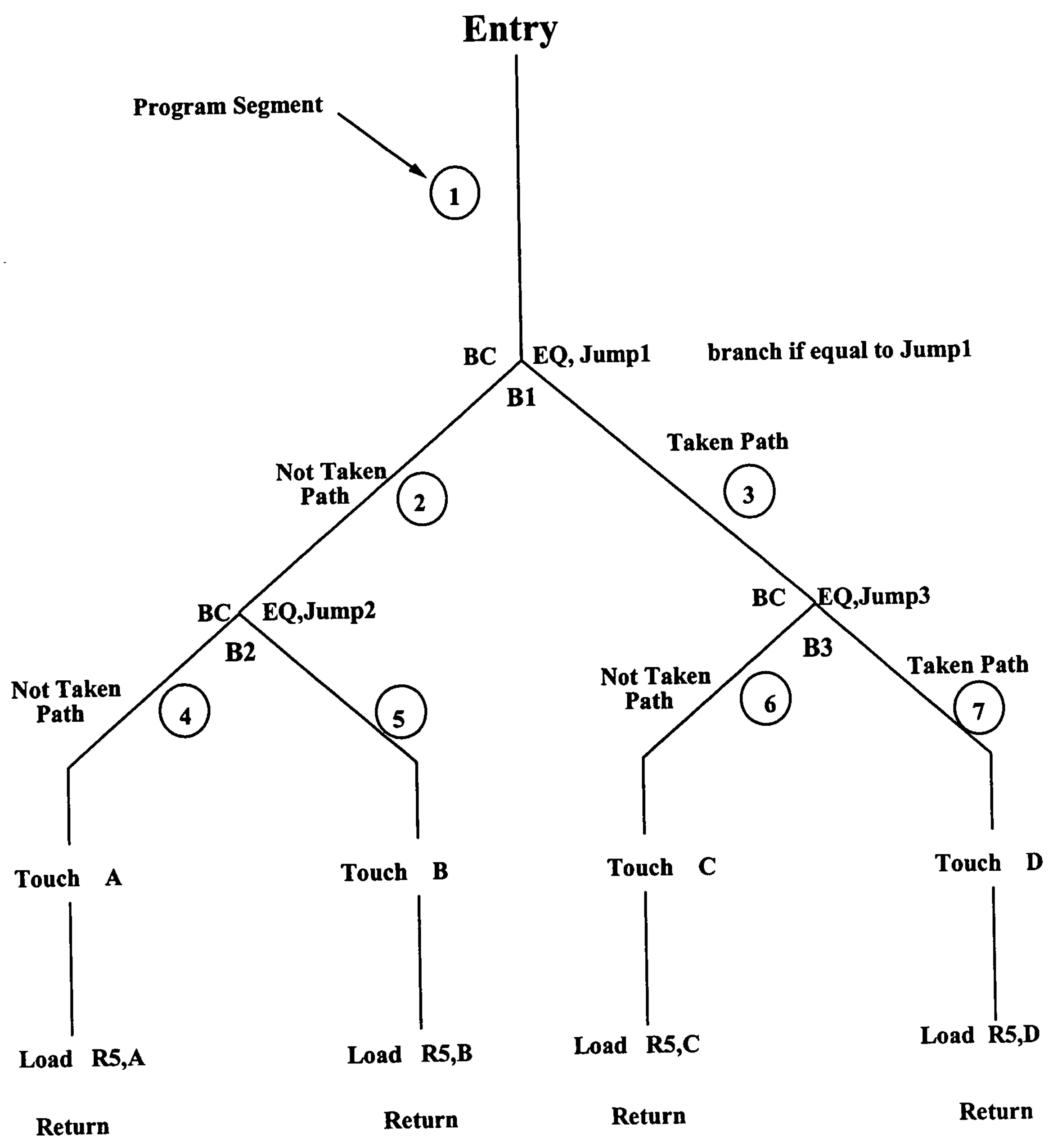


Figure 2

Control Flow Graph Of Program

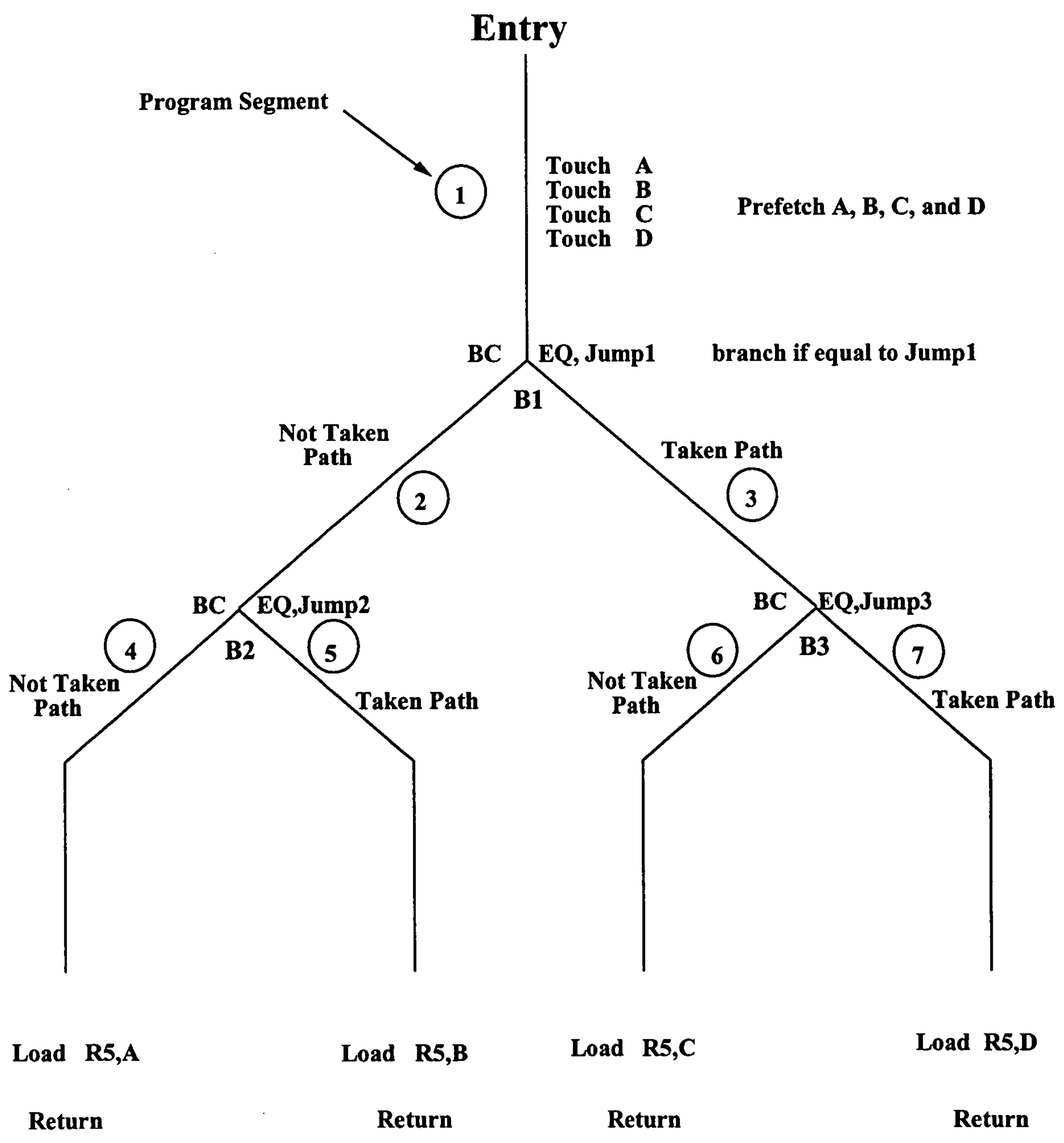
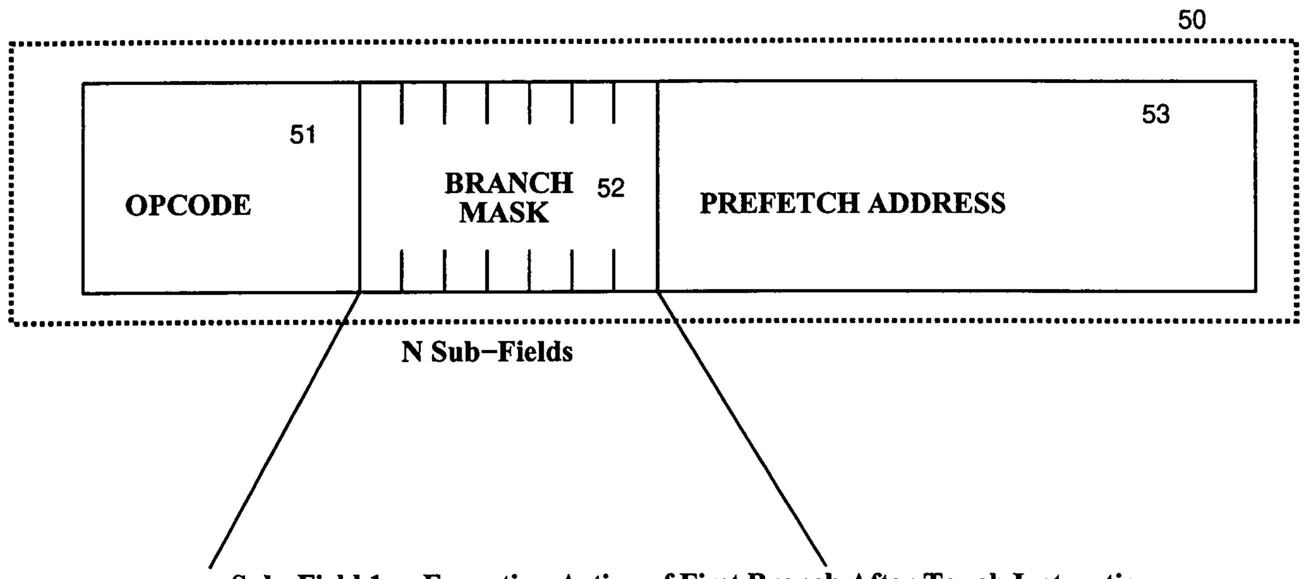


Figure 3

TOUCH INSTRUCTION FORMAT



Sub–Field 1 = Execution Action of First Branch After Touch Instruction

Sub–Field 2 = Execution Action of Second Branch After Touch Instruction

•
•
•

Sub–Field N = Execution Action of Nth Branch After Touch Instruction

The Branch Action Can Be One Of The Following Values:

T	=	The Branch Is Taken
N	=	The Branch Is Not–Taken
D	=	The Branch Can Either Be Taken Or Not–Taken, 'Don't Care'

Prefetch Address Can Denote A Base Register + Displacment or
A Relative Offset From the Touch Instruction

Figure 4

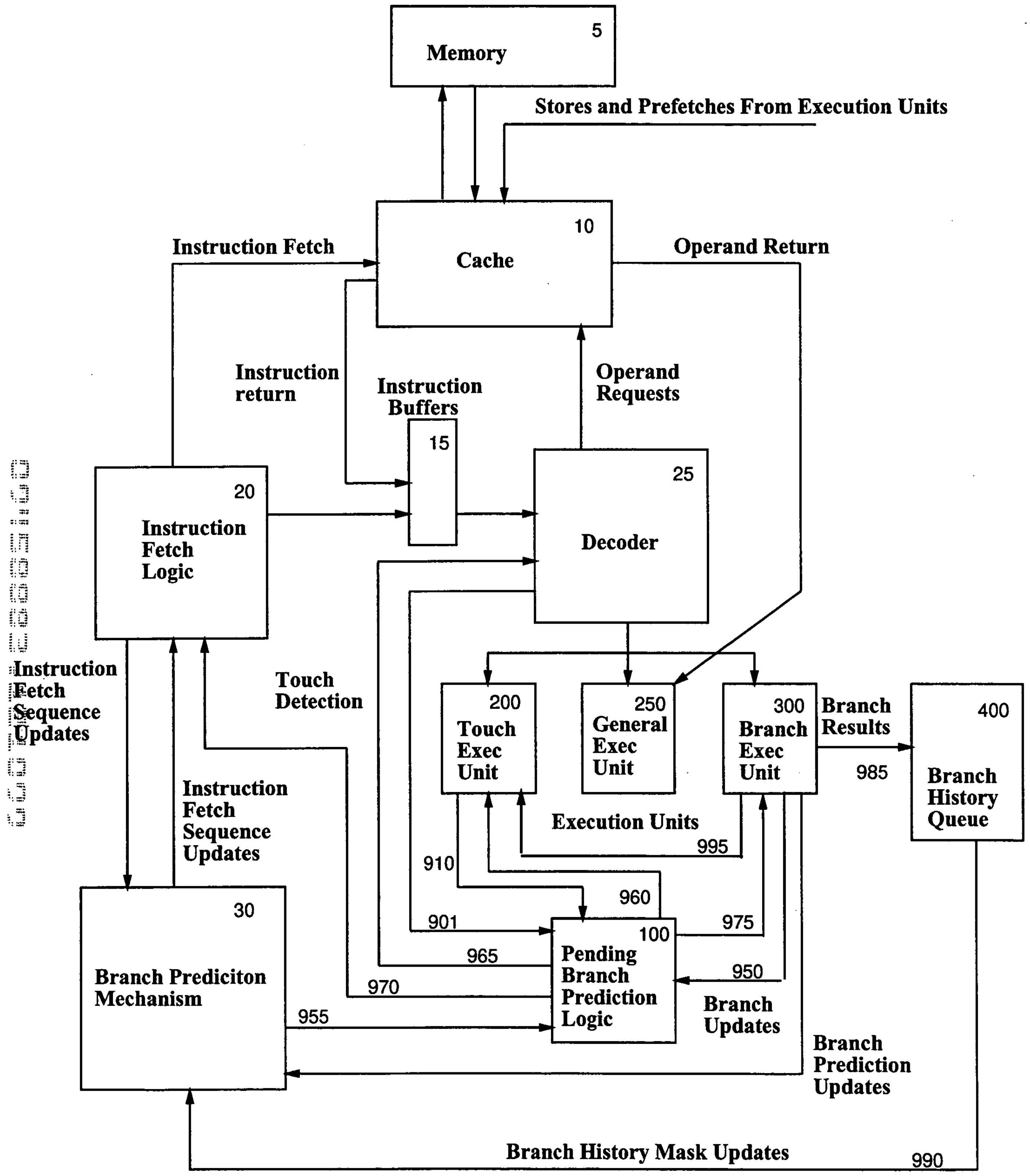
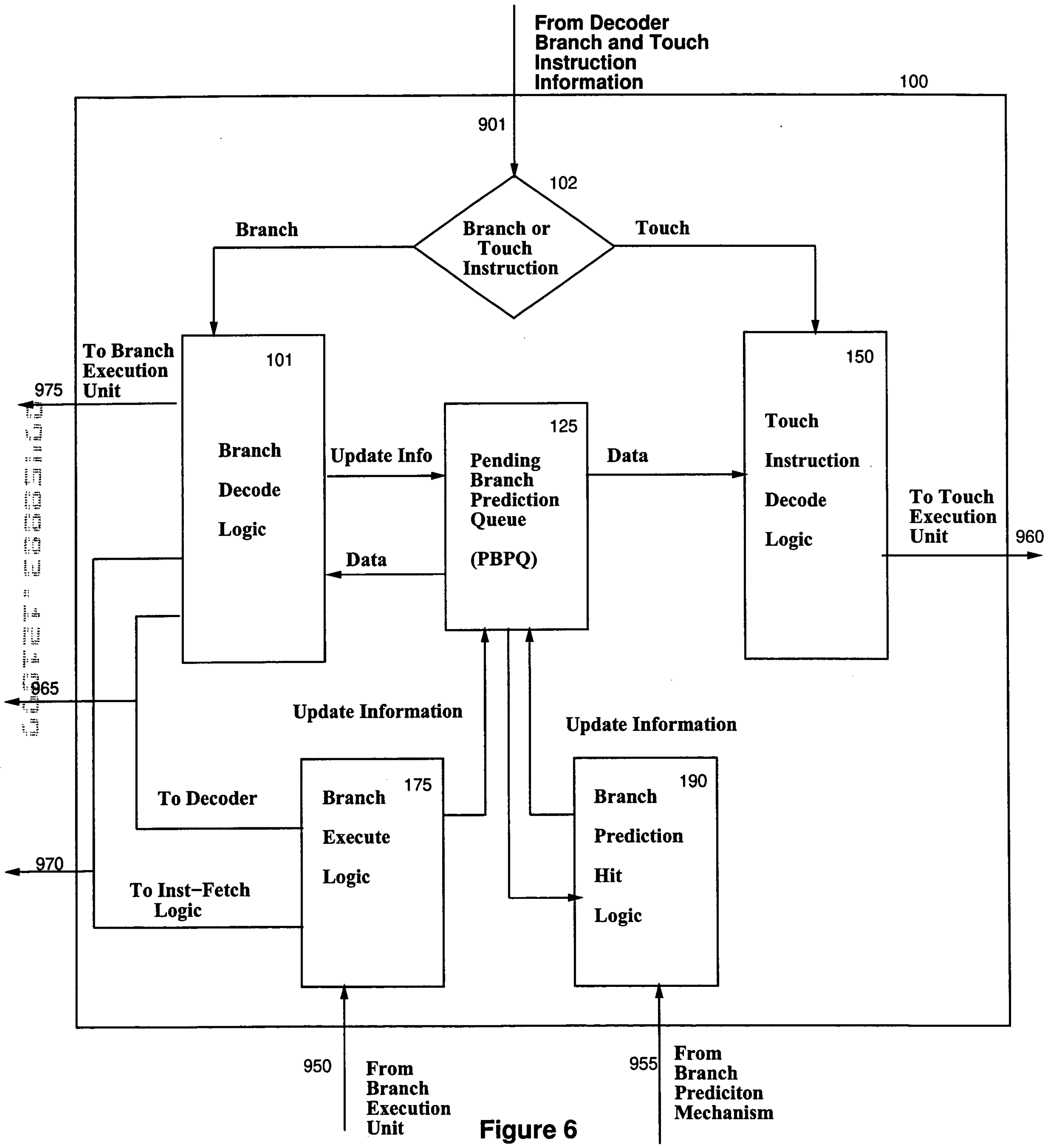

Memory
5
Stores and Prefetches From Execution Units
Cache
10
Instruction Fetch
Operand Return
Instruction return
Instruction Buffers
15
Operand Requests
Decoder
25
Instruction Fetch Logic
20
Instruction Fetch Sequence Updates
Touch Detection
Touch Exec Unit
200
General Exec Unit
250
Branch Exec Unit
300
Branch Results
985
Branch History Queue
400
Execution Units
995
910
960
Instruction Fetch Sequence Updates
Branch Prediciton Mechanism
30
901
975
Pending Branch Prediction Logic
100
965
950
970
Branch Updates
955
Branch Prediction Updates
Branch History Mask Updates
990

Figure 5

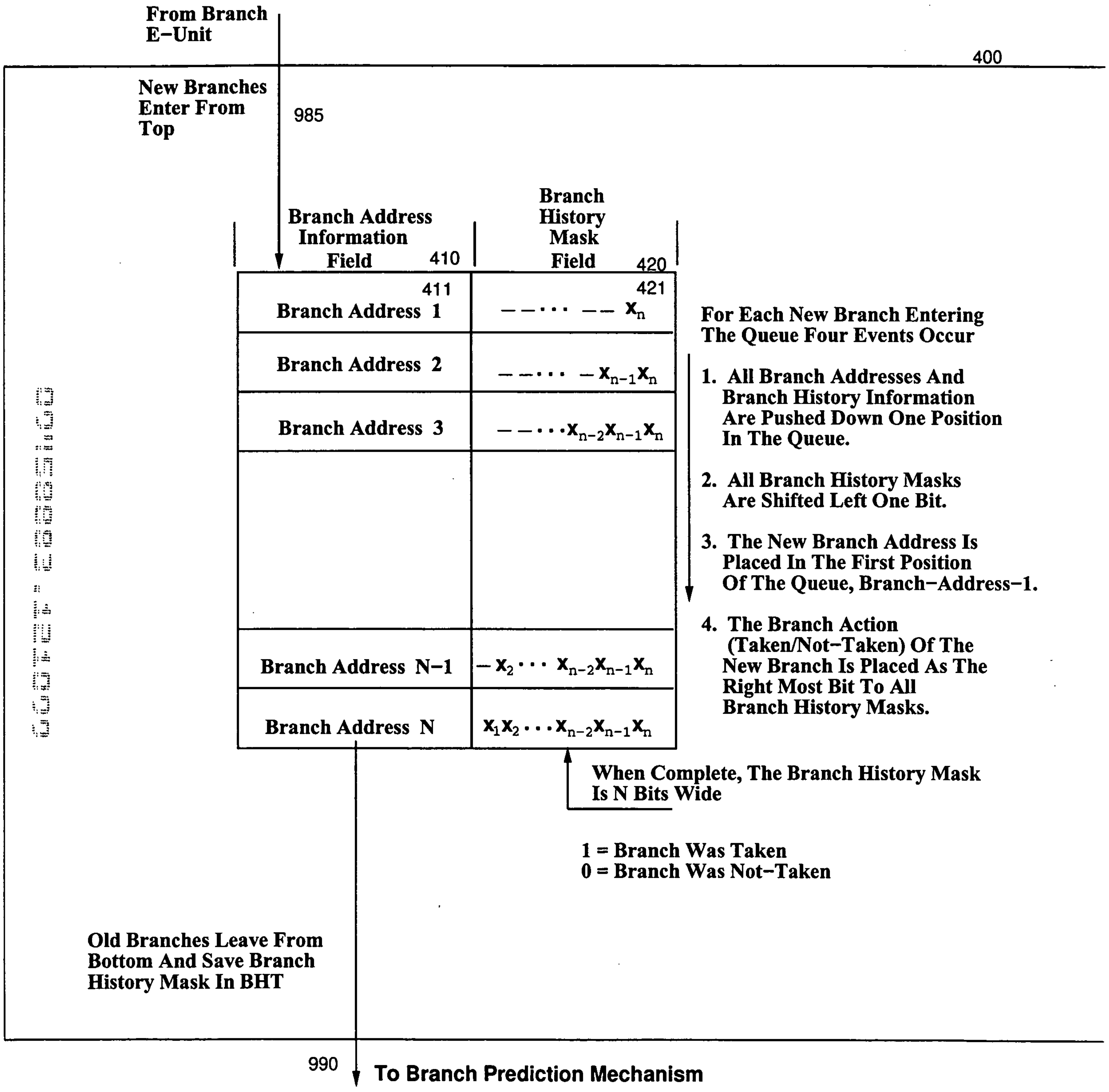
Pending Branch Prediction Logic



From Decoder Branch and Touch Instruction Information
100
901
102
Branch or Touch Instruction
Branch
Touch
To Branch Execution Unit
975
101
Branch Decode Logic
Update Info
125
Pending Branch Prediction Queue (PBPQ)
Data
Data
150
Touch Instruction Decode Logic
To Touch Execution Unit
960
965
Update Information
Update Information
To Decoder
175
Branch Execute Logic
190
Branch Prediction Hit Logic
970
To Inst-Fetch Logic
950
From Branch Execution Unit
955
From Branch Prediciton Mechanism

Figure 6

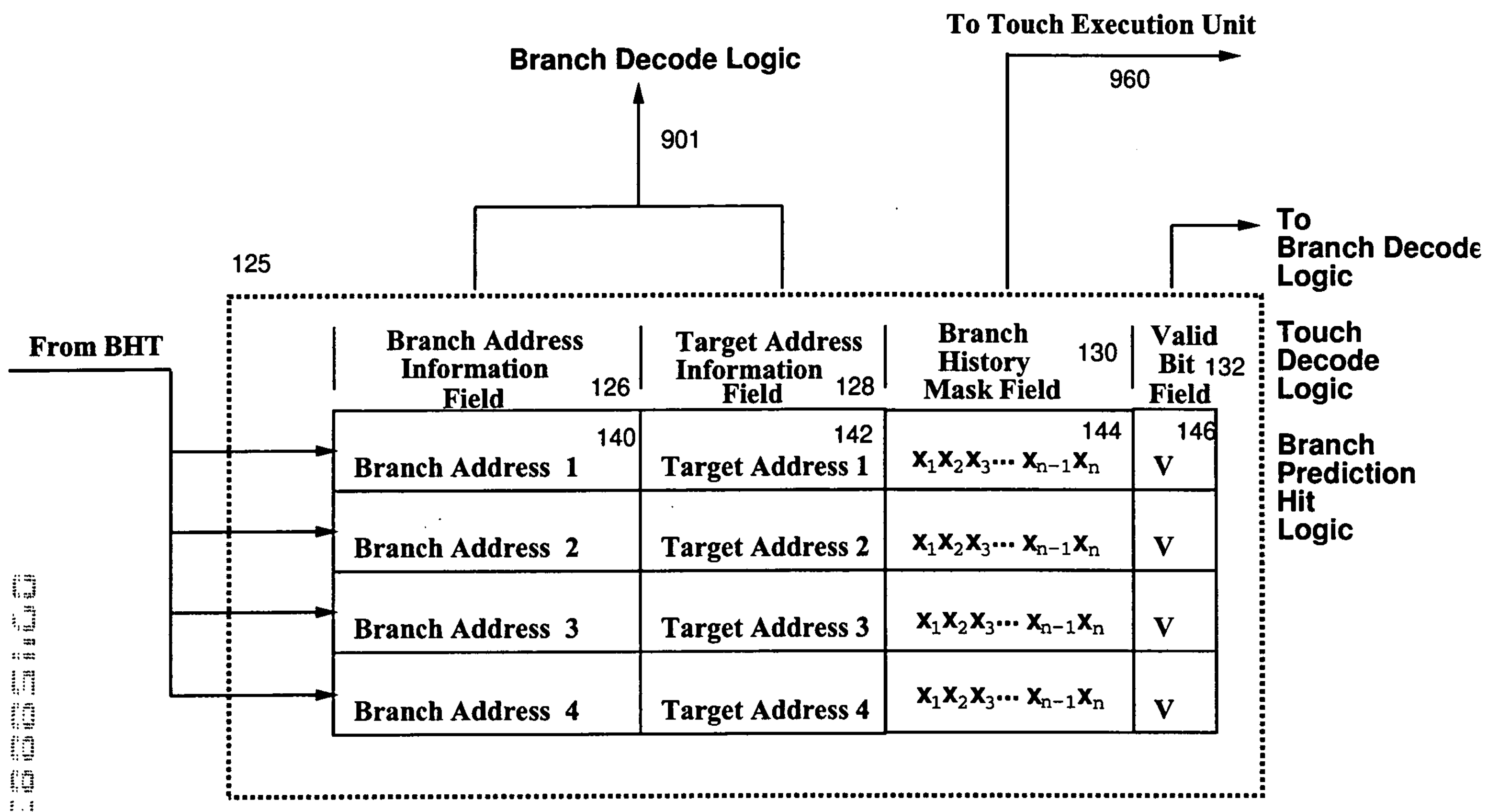
BRANCH HISTORY QUEUE



From Branch E−Unit
400
New Branches Enter From Top
985
Branch Address Information Field
410
Branch History Mask Field
420
411
Branch Address 1
421
$— — \cdots — — x_n$
Branch Address 2
$— — \cdots — x_{n-1}x_n$
Branch Address 3
$— — \cdots x_{n-2}x_{n-1}x_n$
Branch Address N−1
$— x_2 \cdots x_{n-2}x_{n-1}x_n$
Branch Address N
$x_1x_2 \cdots x_{n-2}x_{n-1}x_n$
For Each New Branch Entering The Queue Four Events Occur
1. All Branch Addresses And Branch History Information Are Pushed Down One Position In The Queue.
2. All Branch History Masks Are Shifted Left One Bit.
3. The New Branch Address Is Placed In The First Position Of The Queue, Branch−Address−1.
4. The Branch Action (Taken/Not−Taken) Of The New Branch Is Placed As The Right Most Bit To All Branch History Masks.
When Complete, The Branch History Mask Is N Bits Wide
1 = Branch Was Taken
0 = Branch Was Not−Taken
Old Branches Leave From Bottom And Save Branch History Mask In BHT
990
To Branch Prediction Mechanism

Figure 7

PENDING BRANCH PREDICTION QUEUE



Branch History Mask Records The Branch Action Of Last N Branches
1 = Taken
0 = Not–Taken

Branch Address Information
Address Of Predicted Branch Detected During Branch Prediction

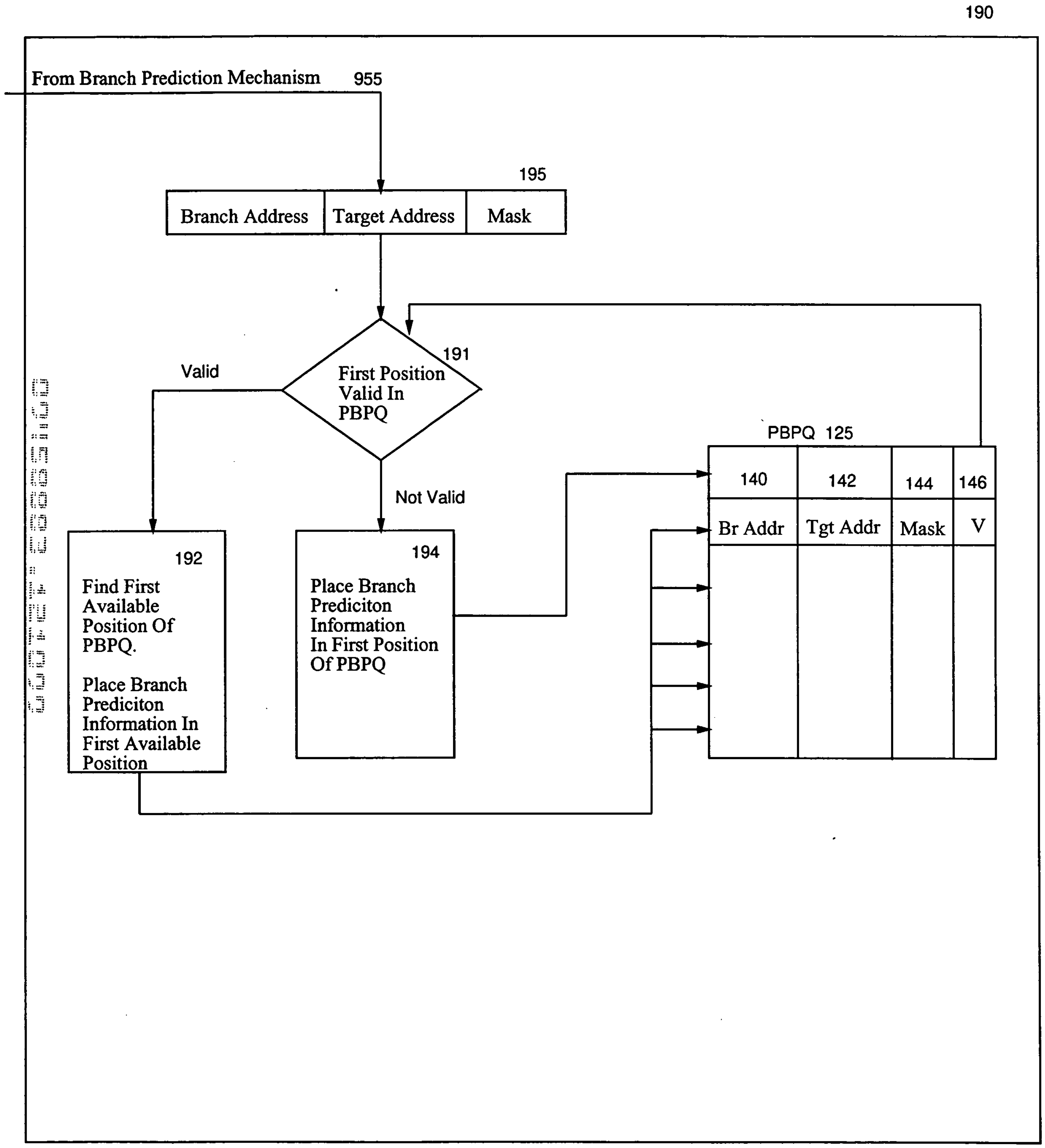
Target Address Information

Predicted Target Address of Branch Detected During Branch Prediction

Valid Bit
1 = Valid
0 = Invalid

Figure 8

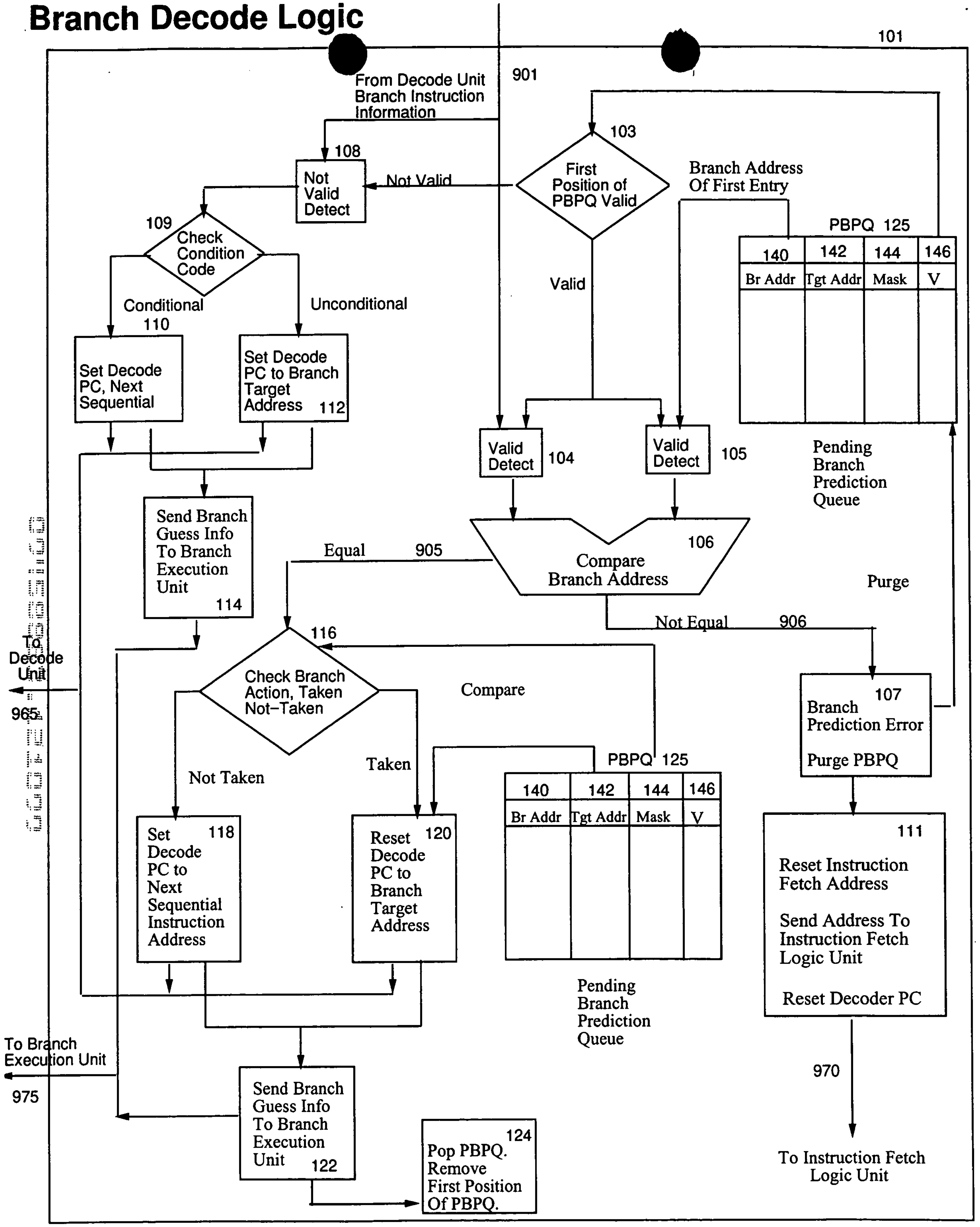
Branch Prediciton Mechanism Hit Logic



190
From Branch Prediction Mechanism
955
195
Branch Address
Target Address
Mask
191
First Position Valid In PBPQ
Valid
Not Valid
192
Find First Available Position Of PBPQ.
Place Branch Prediciton Information In First Available Position
194
Place Branch Prediciton Information In First Position Of PBPQ
PBPQ 125
140
142
144
146
Br Addr
Tgt Addr
Mask
V

Figure 9

Branch Decode Logic



101
From Decode Unit Branch Instruction Information
901
103
First Position of PBPQ Valid
Branch Address Of First Entry
108
Not Valid Detect
Not Valid
109
Check Condition Code
Conditional
110
Unconditional
Set Decode PC, Next Sequential
Set Decode PC to Branch Target Address 112
Valid
PBPQ 125
140 Br Addr
142 Tgt Addr
144 Mask
146 V
Valid Detect 104
Valid Detect 105
Pending Branch Prediction Queue
Send Branch Guess Info To Branch Execution Unit 114
Equal 905
106
Compare Branch Address
Purge
Not Equal 906
To Decode Unit
965
116
Check Branch Action, Taken Not–Taken
Compare
107
Branch Prediction Error
Purge PBPQ
Not Taken
Taken
PBPQ 125
140 Br Addr
142 Tgt Addr
144 Mask
146 V
Set Decode PC to Next Sequential Instruction Address 118
Reset Decode PC to Branch Target Address 120
111
Reset Instruction Fetch Address
Send Address To Instruction Fetch Logic Unit
Reset Decoder PC
Pending Branch Prediction Queue
To Branch Execution Unit
975
970
Send Branch Guess Info To Branch Execution Unit 122
124
Pop PBPQ. Remove First Position Of PBPQ.
To Instruction Fetch Logic Unit

Figure 10

Touch Instruction Decode Logic

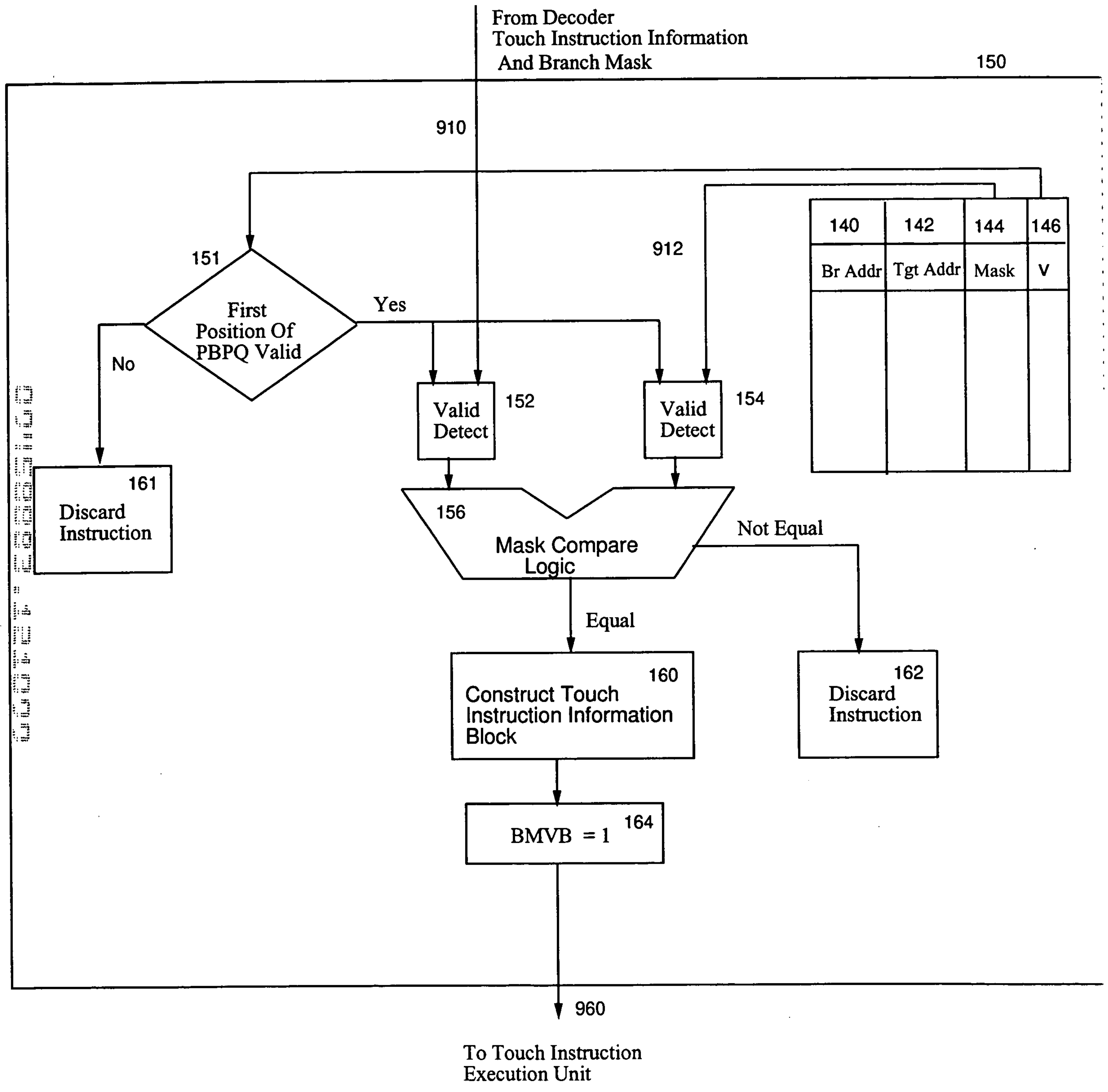


Figure 11

Branch Mask Compare Logic

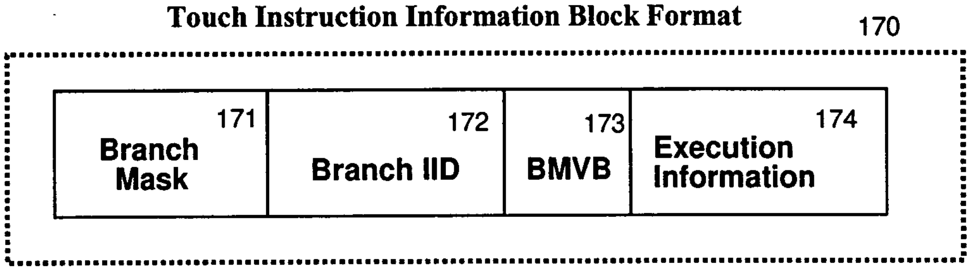
		Input From Touch Instruction Branch Mask		
		T	N	D
Input From Branch History Mask	0	No Match	Match	Match
	1	Match	No Match	Match

0 = Not Taken

1 = Taken

Figure 12

Touch Instruction Information Block Format



Branch Mask = Branch Mask Contained In Touch Instruction

Branch IID = Instruction Identifier Of Last Branch Decoded

BMVB = Branch Mask Validation Bit

Execution Information = Address Of Item To Prefetch, ...

Figure 13

Branch Execute Logic

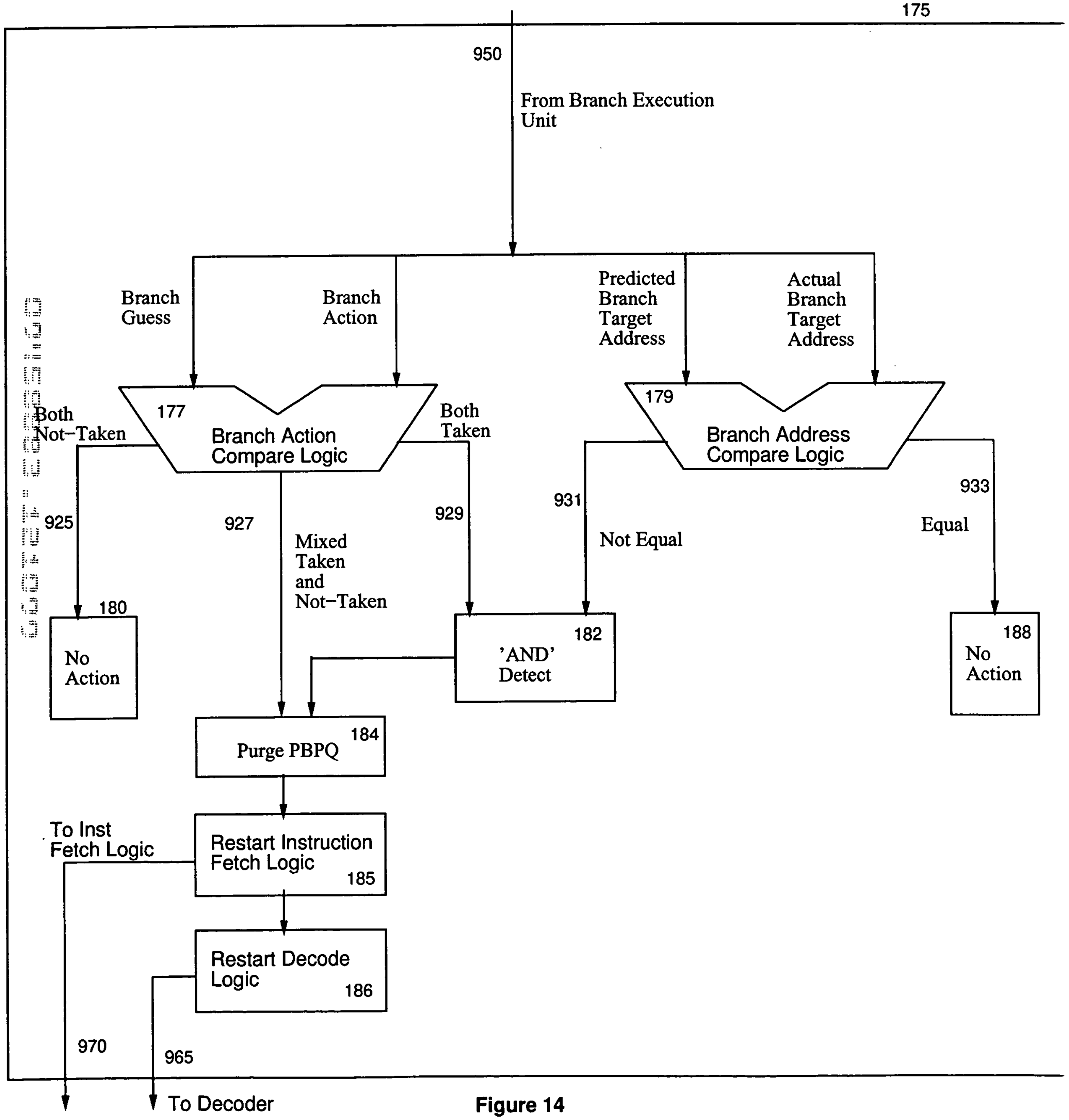


Figure 14

Touch Instruction Execution Unit

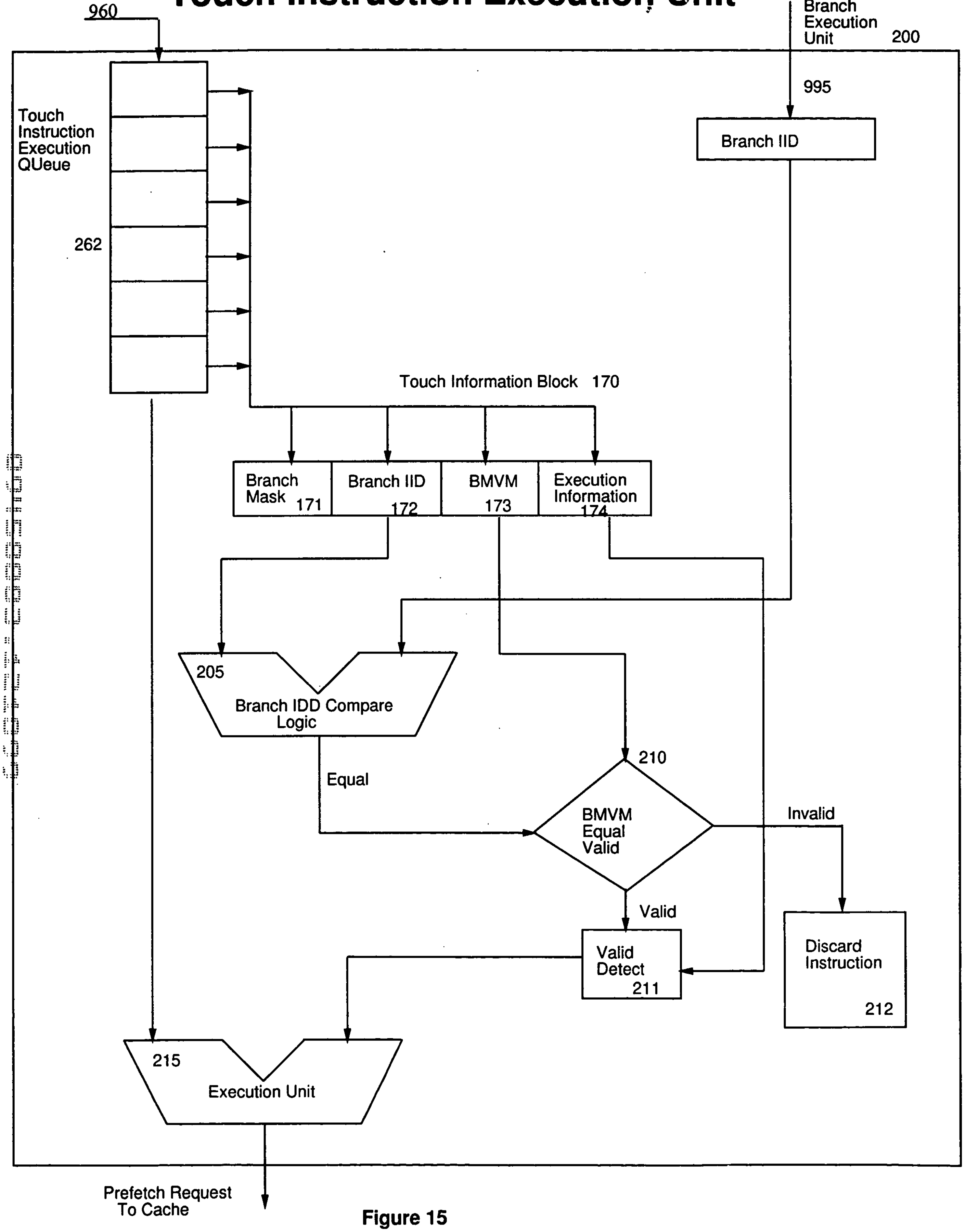


Figure 15